Counting Sort.

// input must be nonnegative int
// A is input, B is output

// for each input A[i], find
// all values less than it.

0) Let MaxVal = max(A);
1) Initialize all C[0 ... MaxVal] to zero
2) For j = 1 to n
   C[A[j]] ++;
   // C[i] is # elements in A equal to i
4) For i = 1 to MaxVal
   C[i] = C[i] + C[i-1].
   // Now C[i] is # elements in A <= i
6) For k = n to 1
   Let key = A[k]. // value in A
   Let count = C[key]. // # elements <= val
   B[count] = key;
   C[key] = count - 1;
Counting sort example

Input A: 2 1 6 5 2 5 3 1

① max Val = max (A) = 6

② Initialize all C[0...6] to ∅

③ Let C[i] = # elements in A equal to i

④ Let C[i] = # elements in A ≤ i
Let $B[1...n]$ be the output array. Note: if $C[i]$ has the # elements $\leq i$,
then $B[C[i]] = i$

Count of all values $\leq i$

Example, if $C[3] = 4$, then there are 4 values $\leq 3$, so put a 3 in the 4th slot in B.
Work backwards from rightmost input element
to leftmost input element.

Input: A: [2, 1, 6, 5, 2, 1, 5, 3, 1]

Scratch: c: [8, 2, 4, 8, 8, 7, 7]

Output: B: [1, 1, 3, 3, 5, 5, 5, 5]

Stable: (sorted)

1 1 1 2 2 2 3 3 3 3 5 5 5 6
Counting Sort

What if we also have negative integers? or $\min \geq \emptyset$?

- compute Min
- compute Max.

- give $C$ size $(\text{Max} - \text{Min})$.
  
  $C[\emptyset \ldots (\text{Max} - \text{Min})].$

- subtract min from all values
- do Counting Sort as before.

- add min to all elements in $B$.

Alternatively, subtract min from values used as indices to $C$. 
Runtime of Counting Sort (array reads, array writes)

\[ \Theta(\max(n, \text{MaxVal})) \]

or

\[ \Theta(\max(n, \text{range})) \]

\[ \frac{\text{max} - \text{min} + 1}{\text{equivalent}} \]

\[ = \Theta(n + \text{range}) \]

If range is \( O(n) \), then algorithm is \( \Theta(n) \).
Radix Sort for nonneg. integers

Sort one digit at a time

329 → 457 → 329 → 329
457 → 657 → 839 → 457
657 → 329 → 657 → 657
839 → 839 → 839

sort
rightmost
digit first

sort
leftmost
digit last

Digit sorts must be STABLE, counting sort is a good choice.

- MaxVal on digit sort is 9,
so digit sort is $O(\max(n,9)) \rightarrow O(n)$.
- So runtime of radix sort is $O(dn)$
  if $n > 9$
General Radix Sort Runtime

d = \# “digits”
  (could be other data)

r = range of each digit

n = \# values.

Radix sort runtime is \( O(d(n+r)) \)

(equivalent to \( O(d \cdot \max(n, r)) \))

if \( d \) is fixed and \( r \in O(n) \),
then radix sort is linear.
**Question 1:** If we have $n$ $b$-bit integers, can we sort them in $\Theta(b \cdot n)$ time?

Use radix sort.

\[
\# \text{ digits} = b \\
\text{range} = 2 \quad (\emptyset \text{ or } 1) \\
n = n
\]

so runtime is $\Theta(\# \text{ digits} (n + \text{ range}))$

$= \Theta(b(n + 2))$

$= \Theta(bn)$
**Question 2**: How many bits are used to represent the numbers in the range $0 \ldots n-1$?

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n-1$</th>
<th># bits needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>4</td>
</tr>
</tbody>
</table>

$\log_2 n$ bits!

So if sorting $n$ $b$-bit integers takes $\Theta(bn)$ time, then sorting $n$ $(\log_2 n)$-bit integers takes $\Theta(n \log n)$ time!
Question 3: What if we group the digits into clusters of size $r$?

\[
\begin{array}{ccc}
2317 & \rightarrow & 2317 \\
6542 & \rightarrow & 4321 \\
4321 & \rightarrow & 1239 \\
0896 & \rightarrow & 6542 \\
1239 & \rightarrow & 0896 \\
\end{array}
\]

$r = 2$

\# "digits" = $2 = 4/r$

range = 100 = $10^2 = 10^r$

so runtime = $\Theta(2(n + 100))$

= $\Theta\left(\frac{4}{r}(n + 10^r)\right)$

If $n$ b-bit digits clustered into clusters of size $r$, runtime is $\Theta\left(\frac{b}{r}(n + 2^r)\right)$
Radix sort one bit at a time is

$$\Theta(b(n+2)) = \Theta(bn)$$

Radix sort $r$ bits at a time is

$$\Theta\left(\frac{b}{r}(n+2^r)\right) = \Theta\left(\frac{b}{r}n\right)$$

Save time, but counting sort now needs more space...
(Max Val is bigger when $r > 1$)

In decimal example, Max Val grew from 9 to 99
Claim: Given the number of bits to represent a number is $O(\log n)$, then if we do all of the bits in a single grouping, Radix Sort runs in $\Theta(n)$ time.

$b = O(\log n)$

Let $r = \log_2 n$ (i.e., # of bits)

Radix Sort is $\Theta\left(\frac{b}{r} (n + 2^r)\right)$ \[ \frac{b}{r} = 1 \]

$\Theta(n + 2^r)$

$\Theta(n + 2^{\log_2 n})$

$\Theta(n + n)$

$\Theta(n)$

Lots of hidden constants + memory.
Can we sort $n$ values that are in the range $\phi \ldots n^2$ in $O(n)$ time?
$n$ values in the range $0 \ldots n^2$

ok, let's try regular radix sort

# digits = $d$, $d \leq n^2$

runtime $= \Theta(d(n+r)) = \Theta(d(n+n^2))$

eeek!

Another way:

- represent all values in binary
- split binary number in two
Example: 2 8 6 18 10 36

<table>
<thead>
<tr>
<th>decimal</th>
<th>binary</th>
<th>( n = 6 )</th>
<th>( n^2 = 36 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>000010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>001000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>000110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>010010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>001010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>100100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# "digits" = 2 (# clusters)
maxVal = 111 (binary) or \( 7 \leq 2n \)
So runtime

\[
\Theta(d(n+r)) = \Theta(2(n + (2n)) = \Theta(n)
\]
Bucket Sort

Linear expected time.

Assume elements equally distributed across the range of values.

Create n buckets.

Example: numbers between $\phi$ (inclusive) and 1 (exclusive) $[\phi, 1)$
- bucket 1: $[\phi \ldots \frac{1}{n})$
- bucket 2: $[\frac{1}{n} \ldots \frac{2}{n})$
- bucket n: $[\frac{n-1}{n} \ldots \frac{n}{n})$

Example: numbers between $\phi$ and 1,000
- bucket 1: $[\phi \ldots \frac{1000}{n})$
- bucket n: $[\frac{n-12}{n} \ldots \frac{1000}{n})$
Bucket Sort (A[i...n]) {

① Determine RANGE of values using MAX and MIN
② Create n equal-sized buckets based on RANGE
③ for i = 1 to n
   insert A[i] into corresponding bucket
④ Sort each bucket
⑤ Concatenate buckets

Runtime: MAX, MIN: O(n)
    put in buckets: O(n)
    concatenate buckets: O(n)
    but sort each bucket?
How long to sort each bucket?

Example: 8 3 2 φ

range: φ → 8

n = 4

Bucket 1   Bucket 2   Bucket 3   Bucket 4
[φ−2)   [2−4)   [4−6)   [6−8]

φ  3, 2  8

sort φ  2, 3  8

answer φ 2 3 8

Note: # elements in each bucket very low!!
b/c of our ASSUMPTION of uniform data values!
b/c of our ASSUMPTION of uniform data values:

expected # of values per bucket is very small.

example: if max bucket size is 8,
then # comparisons to sort = \( \frac{24}{n} \).
sort time is \( \frac{24}{n} \times n \text{ lists} = 24n \).
Total runtime is \( \Theta(n) \).